Response time consistency of the GHOST force loop

A. E. Kirkpatrick and Jason Sze
School of Computing Science
Simon Fraser University
Burnaby, BC, V5A 1S6 Canada
{ted,jszea}@cs.sfu.ca

Abstract

The consistency with which Windows 2000 invokes the GHOST force loop was measured on a 900 MHz machine. The median loop time was accurate at 1 ms. However, as the computation in the force loop increased to 500 ms, the consistency of the loop decreased up to 25%. Inaccuracies in loop times were also introduced by higher network load.

Introduction

The quality of the force display is an important factor in haptic applications[1]. The device, control algorithms, and application program all contribute to the quality of the display. All these components run in cooperation with a silent partner, the operating system kernel. The kernel provides the environment in which the device driver runs and the low-level facilities by which the driver communicates with the hardware. It provides the file system, network, interprocess communication, and virtual memory facilities upon which the application is based. Most importantly, the kernel scheduler determines when the application will refresh the force and graphic displays and the world model.

While much work has been done to locate and eliminate inadequacies in force display, control algorithm, and application design, we are not aware of any research on the effect of the operating system algorithms on the performance of haptic systems. To date, researchers and application programmers alike have presumed that the operating system is "good enough". In this paper, we consider the effects the operating system scheduler might have on this performance. We begin by considering the possible mechanisms by which the scheduler might impact system performance. We point out that an important measure of system performance should be the distribution of response times for the force refresh loop. We then present some initial measurements of the response time distribution for a particular PHANTOM configuration, a single-processor system running Windows 2000 and GHOST 3.1. We conclude with a discussion of the implications of these results for haptic applications.

---

[1] In a talk given at PUG '01, the first author emphasized the importance of separating our terms for display technologies—forces and graphics—from the haptic and visual systems, the human perceptual systems interpreting those displays. In keeping with that principle, we shall refer to "force displays", the "force rendering loop", and so forth. However, given the established usage of such broader terms as "haptic application" and "haptic programming", we retain them here.

The operating system scheduler and the structure of a haptic application

Haptic programming is inherently multi-threaded. The classic structure for haptic applications consists of two loops, one computing the force display based upon the current cursor position and the location of objects in the world model, the second computing the graphic display based upon the same information. Because the human visual and tactual receptors have differing response rates, these loops run at different rates. Currently, SensAble's GHOST software architecture sets the force loop at 1000 Hz. Graphics loops typically run between 10 and 40 Hz. More importantly, because the consistency requirements of the tactual mechanoreceptors are more stringent than for the visual receptors, the force refresh loop is typically run at a higher priority than the graphics loop.

An important but little discussed consequence of this multithreaded architecture is that it makes the operating system an inherent component of the application, with operating system scheduling algorithms limiting the application's quality of service. The application (or, in the case of GHOST users, the application framework) may request a theoretical rate of force display but it is the scheduler that determines the actual rate. This scheduler is itself a complex algorithm, particularly when considered in terms of its interactions with the other services provided by the operating system. Consequently, it is imperative for haptic programmers to have clear descriptions of the limits of the scheduler they are using. In this paper, we begin developing such a description.

The scheduler as a source of noise in the force signal

The fundamental force computation is a read-compute-write loop. For each tic of the clock, the application reads the current location of the cursor, computes forces at the tip based upon its interactions with nearby objects, then writes the forces back to the motors of the display. The scheduler determines when each iteration of the loop occurs.

There are two possible kinds of noise the scheduler can introduce into the force signal. First, it can invoke the force calculation consistently slower or faster than the stated rate. We refer to this effect as *drift*. Second, the time between successive invocations may vary. We call this effect *spread*. We note that some degree of drift and spread is inevitable. The important question is their magnitude.

Spread and drift have several consequences. Irregular invocations of the force loop can create irregularities in changes to the displayed force. If the application assumes time between loop iterations is constant, timing spread will produce inaccurate force computations. This is potentially most severe for algorithms that use higher-order terms such as velocity and acceleration. Longer delays between force computations can produce large force changes when the cursor is penetrating a rigid surface. Extreme force changes can cause the drivers to shut down the computation. Finally, irregular force computations can produce inaccurate high-frequency forces. In the worst case, small surface features may be omitted altogether.

The ultimate metric of the above effects is psychophysical: Does the human user perceive the irregularities in a given application? However, such a metric incorporates far more than the operating system effects. For the purpose of measuring the specific contribution of operating system delay, we instead directly timed the invocations of the force loop.

Method

Timing loop consistency was measured using a skeletal GHOST application. The virtual world consisted of a single object located at the origin of the coordinate system. The object had a bounding volume far larger than the PHANTOM working area, causing its collisionDetect () method to be called for every invocation of the force loop. The collisionDetect () routine stored the time of its invocation in an array. Time was recorded using the Windows HighPerformanceTimer facility. This timer operates under ten μs accuracy, more than good enough for accurate measurement of the one ms force loop. The collisionDetect () routine optionally executed a busy delay loop. The delay simulated varying degrees of force computation. The collisionDetect () routine completed by returning zero, indicating no collision with the object.

The body of the application consisted of a graphics loop executed by a Windows timer interrupt every ten ms. This loop also recorded its execution time but this data is not reported here. The graphics loop displayed the PHANTOM cursor as an OpenGL sphere but did no other rendering.

This sample application was run for one minute. The PHANTOM was not touched during this time. As there were no objects for the tip to contact in this virtual world, moving the PHANTOM would have had no effect on the results. After the minute elapsed, the differences between the 60,000 successive force loop invocations were computed and written to a file.

Timings were collected under various conditions. In the unloaded condition, no other applications were active. In successive runs, the force loop delay was varied from 0 to 700 μs in increments of 100 μs, with a final run of 750 μs. In the loaded condition, a network-intensive application was run simultaneously. The network application opened up ten TCP connections and received an 888 byte message from each connection every ms, for a total throughput of 8.9 Mb/s. This application created conditions of high network load. The GHOST timing application was run concurrently, again with force loop delays ranging from 0 to 750 μs.

All timing runs were performed on a Windows 2000 (release 5.00.2195, SP2) system with GHOST 3.1. The hardware was a single-processor AMD 900 MHz Thunderbird with 500 Mb of PC-133 SDRAM and an nVidia GeForce2 card.

Results

To measure the extent of drift and spread, the .01, .05, .50, .95, and .99 quantiles were computed for the 59,999 elapsed loop times for each run. For all runs, the median (.50 quantile) was 1.00 ms. This configuration of Windows has effectively no drift under the tested conditions.

To .01, .05, .95, and .99 quantiles are a good measure of the spread of the loop times. The inner .05 to .95 band indicates the range of 90% of the loop times. A further 8% of the loop times lie in the outer band, outside the .05 -.95 range but within the .01 and .99 quantiles. Given a uniform distribution of these times, on average one loop every 80 ms (12.5 Hz) would fall in the outer 8% band. This frequency is within the detection range of human mechanoreceptors.

Figure 1 shows a plot of the .01, .05, .95, and .99 quantiles for the unloaded condition (solid lines) and the loaded condition (dashed lines). The 90% band is tight, ±.03 ms. While the 90% band is constant irrespective of the loop delay, the 95% band widens as the computation load of the haptic loop increases, from ±.08 ms at the 0 ms delay to a negatively skewed interval (0.66 to

1.18 ms) at the 500 ms delay. Beyond 500 ms, the lower limit of the loop time becomes bounded from below by the loop delay itself and the range shrinks, although the upper limit remains high.

The loaded condition has a wider spread than the unloaded condition but is much less affected by increasing loop delay. The 90% band is wider (±.08 ms) than for the unloaded condition. The 95% band is relatively constant with a more symmetric interval (.72 to 1.21 ms) of about the same width as the unloaded condition. Unlike the 95% band for the unloaded condition, however, the loaded band is basically constant across the range of delays, with only a slight decrease in the lower limit.

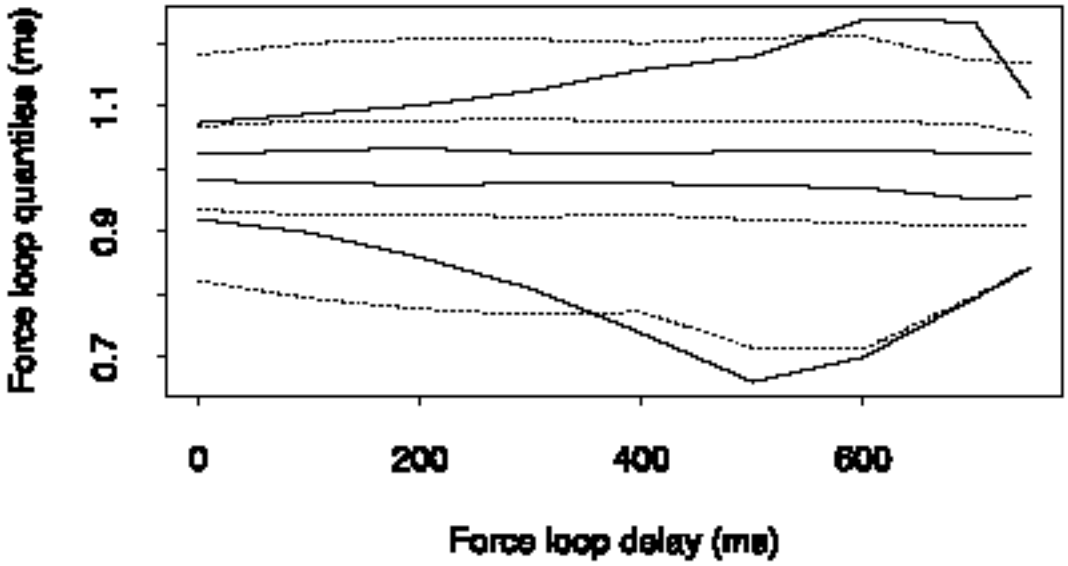

Figure 1: .01 (lowest line), .05, .95, and .99 (highest line) quantiles for unloaded (solid lines) and loaded (dashed lines) conditions at various levels of delay.

Conclusions

This system configuration exhibited generally stable performance in the consistency of its force loop. However, even for conditions of minimal load, there was sufficient spread to the response times to have humanly perceptible effects. Developers of haptic applications with exacting performance standards may wish to account for instabilities introduced by the operating system scheduling algorithms. Of course, the range of haptic configurations is vast. The operating system and GHOST release used in this initial study have both been superceded by more recent versions. Multiprocessor configurations will show less direct interference but synchronization of memory accesses between the processors may introduce new problems. In general, we suggest that the operating system scheduler will have potential impact on the performance of haptic applications for typical configurations of the foreseeable future. We plan to extend this work to measure the performance of more elaborate configurations. We also plan to examine the psychophysical consequences of the variability introduced by the operating system.

Acknowledgements

Funding for this work was generously provided by NSERC and a Simon Fraser University President's Research Grant.